UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

18|19

Trabajo fin de grado

Herramienta de carga, integración y procesado de datos abiertos

David Paarup Peláez

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

Campus Internacional excelencia UAM CSIC

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Herramienta de carga, integración y procesado de datos abiertos

Validación con el catálogo de datos del Ayuntamiento de Madrid

Autor: David Paarup Peláez
Tutor: Iván Cantador Gutiérrez

mayo 2019

David Paarup Peláez
Herramienta de carga, integración y procesado de datos abiertos

David Paarup Peláez
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

Resumen

Los datos abiertos son datos accesibles, generalmente de forma online, y reutilizables, sin exigencia de permisos específicos ni restricciones de derechos de autor o patentes.

En los últimos años, entre otras, entidades gubernamentales han mostrado especial interés y actividad en la publicación de este tipo de datos. Por ejemplo, Madrid publica las actuaciones de su cuerpo policial, Copenhague, su emisión de CO2, EEUU, sus préstamos a estudiantes universitarios, Bilbao, sus indicadores económico-financieros y Estocolmo, sus aparcamientos para bicicletas. La creciente demanda de transparencia a los gobiernos implica un aumento del número de publicaciones de datos abiertos, con el consiguiente incremento en la variedad de formatos disponibles.

Al estar los datos presentes en diversos formatos, surge la necesidad de desarrollar herramientas capaces de procesarlos e integrarlos. En este TFG se propone desarrollar Balloon, un sistema informático que aúna datos procedentes de diferentes fuentes en Sistemas de Gestión de Bases de Datos, de forma completamente automática.

Para validar el sistema, se realiza la carga de un gran número de conjuntos de datos abiertos heterogéneos del Ayuntamiento de Madrid, con origen en diferentes formatos. Ejemplos de ellos son el callejero, las incidencias de tráfico en vía pública y las actuaciones en materia de limpieza urbana.

Además, como caso de uso, se emplea el sistema para procesar la colección de datos abiertos de Decide Madrid, la plataforma electrónica de presupuestos participativos de la ciudad. Posteriormente, se lleva a cabo un proceso integrador de cruce entre éste y el resto de conjuntos procesados. Finalmente, con el objetivo de proporcionar un contexto objetivo a las propuestas participativas, se analizan los conjuntos integrados y se presentan varios estudios breves relacionados con las propuestas ciudadanas y su intersección con los datos abiertos publicados por el Ayuntamiento.

Palabras clave

datos abiertos, minería de datos, gobierno abierto, transparencia, elasticsearch, kibana, mariadb, java, madrid, decide madrid

Abstract

Open Data is accessible and reusable data, mostly provided online, which are not subject to specific permits, author rights restrictions or patents.

In the last years, among others, governmental entities have shown special interest and activity regarding the publication of this type of data. For instance, Madrid publishes online data about police operations, Copenhagen reports its CO2 emissions, USA publishes its student loans, Bilbao reports financial and economic indicators, and Stockholm provides information about bike parking. In this context, the increasing demand for governmental transparency implies a growing number of Open Data publications, with the consequent increase in the variety of available data formats.

As the data is presented in several formats, tools for processing and integrating them need to be developed. Addressing this situation, this work proposes the development of Balloon, an information system that automatically loads, processes and integrates Open Data from different sources, and stores them into Database Management Systems.

To assess the system, we tested it with a wide number of heterogeneous datasets provided in different formats by Madrid's City Council. Examples of these datasets are the city street map, its traffic incidents' reports, and its urban cleaning operations.

Moreover, as an application case of the system, we also used the system to process the Open Data collection of Decide Madrid, the city's participatory budgeting e-platform. We then conduct a mapping process that links the collection with the remainder datasets. Finally, aiming to provide an objective context for the generated participative proposals, we analyze the integrated datasets and present several valuable, brief studies about the citizens' proposals in the platform according to Open Data published by the city council.

Keywords

open data, data mining, open government, transparency, elasticsearch, kibana, mariadb, java, madrid, decide madrid

Índice

Listas

Lista de códigos

Lista de figuras

Lista de tablas

1

Introducción

Para comprender el término *datos abiertos* es necesario definir la palabra *abierto* en este contexto. La referencia en esta materia es The Open Definition [1]. Este proyecto de la Open Knowledge International [2] publicó la Open Definition 2.1 [3], donde se especifica que:

"El conocimiento es abierto si cualquiera es libre de acceder a él, usarlo, modificarlo y compartirlo, estando sujeto a lo sumo a medidas que preserven su autoría y su apertura.".

Con *medidas que preserven su autoría y su apertura* se refiere a licencias abiertas, es decir, licencias que deben de manera irrevocable permitir el uso, redistribución, modificación, separación, compilación, no discriminación, propagación y aplicación para cualquier propósito de los datos, así como garantizar la gratuidad de estos.

Además, existen las siguientes restricciones:

- Los datos han de encontrarse disponibles como una entidad, descargable a través de Internet.
- Han de ser legibles por una máquina.
- El formato ha de ser abierto. Ejemplos son CSV, JSON, XML y XLSX.

Para comprender el contexto del proyecto también es necesario familiarizarse con el término *Datos Enlazados*, que se refiere a un conjunto de buenas prácticas para publicar y conectar datos estructurados en la Web [4]. Cumplen los siguientes cuatro principios [5]:

- URIs como nomenclatura.
- Empleo de URIs HTTP para que los datos puedan ser encontrados.
- Cuando una URI es accedida, debe proporcionar información útil empleando estándares como RDF y SPARQL.
- Deben proveer URIs a otros datos.

Su objetivo es expandir la Web Semántica [6] para potenciar el procesamiento de datos por parte de personas y máquinas.

Existen múltiples ámbitos para los datos abiertos, que además pueden estar enlazados. Por ejemplo, hay datos de ONGs (UNICEF [7] y WWF [8]), de bancos (Banco Mundial [9]) y de organizaciones gubernamentales. Este proyecto se centra en estos últimos.

Los Datos Abiertos Gubernamentales son publicados por ciudades, países y organizaciones internacionales. Ejemplos de ciudades son:

- Madrid [10].
- Copenhague [11].
- Estocolmo [12].

De países:

- Costa Rica [13].
- Rusia [14].
- EEUU [15].

De organizaciones gubernamentales internacionales:

- Naciones Unidas [16].
- Unión Europea [17].

Con estas publicaciones las entidades cobran transparencia, que es uno de los tres pilares del gobierno abierto. Los otros dos son colaboración y participación [18]:

Figura 1.1: Principios del Gobierno Abierto.

Debido a la gran cantidad de datos y la extensa variedad de formatos en los que se presentan, surge la necesidad de desarrollar software capaz de procesarlos e integrarlos. En este TFG se propone Balloon, un sistema informático que automáticamente aúna los datos procedentes de diferentes fuentes en Sistemas de Gestión de Bases de Datos (SGBD).

La filosofía de Balloon es minimizar el tiempo y el número de acciones que un usuario ha de realizar desde que siente interés por un conjunto de datos hasta que se encuentra presente en un SGBD para realizar consultas.

Para validar el sistema se propone la carga de 15 conjuntos de datos abiertos de la ciudad de Madrid [10].

Figura 1.2: Catálogo de Datos Abiertos del Ayuntamiento de Madrid.

El Ayuntamiento de Madrid ha publicado más de 387 conjuntos que pueden obtenerse de dos maneras:

- Descargando los ficheros. Balloon se centra en estos. En la figura 1.2 se muestra una captura del catálogo de conjuntos de datos del Ayuntamiento.
- Consumiendo la API REST.

Uno puede filtrar los conjuntos por:

- Texto contenido en el nombre o la descripción.
- Formato.
- Frecuencia de actualización (diaria, semanal, mensual...).
- Categoría (deporte, salud, urbanismo, comercio...).

Según el conjunto de datos deseado, los formatos disponibles varían. Ejemplos de conjuntos procesados son el callejero, las incidencias de tráfico en vía pública y las actuaciones en materia de limpieza urbana. La lista completa puede consultarse en la sección 5.2. Se ha tratado de recolectar conjuntos lo más heterogéneos posible para ofrecer un estudio intersante. Los formatos presentes en estos 15 conjuntos son CSV, JSON, XML, RDF y XLSX. Se proporcionan ejemplos reales simplificados en los códigos del 1.1 al 1.4 y en la figura 1.3.

Como caso de uso se presenta el cruce entre el conjunto de *Participación ciudadana. Debates y*

Código 1.1: Ejemplo de registro CSV del conjunto de Divisiones administrativas: distritos, barrios y divisiones históricas.

```
"Codigo_de_barrio";"Codigo_de_distrito_al_que_pertenece";"Nombre_de_barrio";"Nombre_acentuado_del_
    barrio";"Superficie_(m2)";"Perimetro_(m)"
"01";"01";"PALACIO";"PALACIO";"001471085";"005754"
```

Código 1.2: Ejemplo simplificado de registro JSON del conjunto de Actividades Culturales y de Ocio Municipal en los próximos 100 días.

```
{
  "id": "10920062",
  "title": "30_ADH._Artículos_de_los_derechos_humanos",
  "description": "Viernes_5_y_viernes_12:_a_las...",
  "price": 1,
  "dtstart": "2019-04-05_00:00:00.0",
  "dtend": "2019-04-21_23:59:00.0",
}
```

Código 1.3: Ejemplo simplificado de registro XML del conjunto EMT. Indicencias del servicio.

```
<item>
  <guid>78BDC3D4-B259-43FC-A7DD-5D34983C8946</guid>
  <title>Concentración: Avenida Menéndez Pelayo.</title>
  <description>El 10 de marzo de 12:00...</description>
  <author>cgi@emtmadrid.es</author>
</item>
```

Código 1.4: Ejemplo simplificado de registro RDF del conjunto de Actividades gratuitas en Bibliotecas Municipales en los próximos 60 días.

```
<c:Vevent>
  <c:uid>10896574</c:uid>
  <v:fn>2 de Mayo de 1808</v:fn>
  <c:dtstart>2019-05-14 11:00:00.0</c:dtstart>
  <c:dtend>2019-05-14 23:59:00.0</c:dtend>
  <c:summary/>
</c:Vevent>
```

	A	B	C	D
1	FECHA	RANGO HORARIO	DIA SEMANA	DISTRITO
2	01-01-2018	DE 00:00 A 00:59	LUNES	USERA
3	01-01-2018	DE 00:00 A 00:59	LUNES	USERA
4	01-01-2018	DE 00:00 A 00:59	LUNES	USERA
5	01-01-2018	DE 1:00 A 1:59	LUNES	HORTALEZA
6	01-01-2018	DE 1:00 A 1:59	LUNES	HORTALEZA
7	01-01-2018	DE 1:00 A 1:59	LUNES	HORTALEZA
8	01-01-2018	DE 1:00 A 1:59	LUNES	HORTALEZA
9	01-01-2018	DE 1:00 A 1:59	LUNES	HORTALEZA
10	01-01-2018	DE 1:00 A 1:59	LUNES	HORTALEZA

Figura 1.3: Ejemplo simplificado de registro XLSX del conjunto de Accidentes de tráfico de la Ciudad de Madrid de 2018.

propuestas (de ahora en adelante conjunto de Decide Madrid) en los Datos Abiertos con el resto de conjuntos procesados.

Decide Madrid es el lugar donde los ciudadanos madrileños proponen y discuten soluciones a problemas locales. En algunos casos, se echa en falta un contexto más objetivo para estas propuestas y debates. La motivación del caso de uso es proveer este contexto para que la asignación de recursos por el Ayuntamiento de Madrid sea más eficiente, resultando en la satisfacción ciudadana. El resultado que se persigue es una representación gráfica visualmente potente e indispensable para hacer llegar el contexto a los ciudadanos.

La idea es que este trabajo, aunque validado con este caso de uso, sea extensible a cualquier fuente de datos y circunstancia, por lo que el enfoque es muy genérico. Es capaz de procesar cualquier conjunto de datos abiertos en los formatos arriba citados. Basta con un fichero de descripción de los datos y Balloon se encarga de importarlos en el SGBD correspondiente, ocupándose de manera completamente automática de todos los procesos asociados, como la creación de tablas e índices. Un aspecto importante que impulsa su genericidad es la extremadamente sencilla extensión en la lectura y procesamiento de otros formatos que se deseasen incorporar.

La estructuración del cuerpo del documento es la siguiente:

1.– En el apartado de *Diseño* se presentan los requisitos del proyecto, se expone la arquitectura del programa desarrollado, la de diversos ficheros descripción y se explica la organización del proyecto.

2.– En el apartado de *Implementación* se discuten algunas cuestiones técnicas de interés referentes al código.

3.– En el apartado de *Evaluación* se incluyen las pruebas que se han efectuado sobre el programa para garantizar su correcto funcionamiento, así como el caso de uso.

4.– Finalmente, se explican las conclusiones obtenidas con la realización del proyecto.

2

Tecnologías empleadas

2.1. Proyecto

- **Elasticsearch 6.7**: SGBD no relacional y distribuido, basado en REST.
  https://www.elastic.co/products/elasticsearch
  Se emplea en el proyecto para integrar los conjuntos de datos procesados y acceder a las visualizaciones que ofrece Kibana.
- **JDK 8u201**: Entorno de desarrollo para Java 8.
  https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
  Se emplea en el proyecto para compilar y ejecutar el código, así como para generar la documentación y empaquetar el programa en un JAR, todo indirectamente a través de Maven.
- **Kibana 6.7**: motor de consulta y visualización de datos procedentes de Elasticsearch.
  https://www.elastic.co/products/kibana)
  Se emplea en el proyecto para generar las visualizaciones gráficamente potentes con los resultados de los análisis.
- **Logstash 6.7**: motor que importa datos de distintas fuentes, los transforma y exporta a varios destinos.
  https://www.elastic.co/products/logstash
  En el proyecto se ha empleado para migrar datos de MariaDB a Elasticsearch.
- **MariaDB 10.3.13**: SGBD relacional basado en SQL.
  https://mariadb.org/
  En el proyecto se ha empleado para integrar los conjuntos de datos procesados.
- **Maven 3.6.0**: programa de gestión de proyectos basado en un fichero de configuración (POM).
  https://maven.apache.org/
  En el proyecto se ha empleado para resolver las dependencias, compilar, ejecutar el programa y las pruebas, generar la documentación y empaquetar Balloon en un JAR.

2.2. Programa

- **commons-io 2.6**: librería de Apache para facilitar el desarrollo de funcionalidad entrada/salida.
  https://mvnrepository.com/artifact/commons-io/commons-io
  En el proyecto se ha empleado para acceder a la clase FilenameUtils, que simplifica el trabajo con cadenas que contienen rutas de ficheros.
- **elasticsearch-rest-high-level-client 6.7**: conector que comunica Java con Elasticsearch.
  https://mvnrepository.com/artifact/org.elasticsearch.client/elasticsearch-rest-high-level-client

En el proyecto se ha empleado en la creación y borrado de índices, así como en la inserción de documentos.

- **java-getopt 1.0.13**: librería de parseo de argumentos de línea de comandos.
  https://mvnrepository.com/artifact/gnu.getopt/java-getopt
  En el proyecto se ha empleado para parsear los argumentos en la llamada al programa.
- **json 20180813**: librería para parsear ficheros JSON.
  https://mvnrepository.com/artifact/org.json/json
  En el proyecto se ha empleado para parsear los ficheros de datos en formato JSON.
- **junit-jupiter-engine 5.5.0-M1**: librería que facilita el desarrollo de tests automáticos.
  https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-engine
  En el proyecto se ha empleado para desarrollar las pruebas que garantizan el correcto funcionamiento del programa.
- **log4j-core 2.11.2**: sistema de logging de Apache.
  https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-core
  En el proyecto se ha empleado para permitir al usuario la decisión sobre qué tipo de información mostrar por pantalla y para guardar esta en ficheros.
- **mariadb-java-client 2.4.1**: conector que comunica Java con MariaDB.
  https://mvnrepository.com/artifact/org.mariadb.jdbc/mariadb-java-client/2.4.1
  En el proyecto se ha empleado en la creación y borrado de tablas, así como en la inserción de registros.
- **poi-ooxml 4.0.1**: librería de Apache que facilita el trabajo con documentos de Microsoft.
  https://mvnrepository.com/artifact/org.apache.poi/poi-ooxml
  En el proyecto se ha empleado para parsear los ficheros de datos en formato XLSX.

3 Diseño

3.1. Requisitos

El proyecto cuenta con los siguientes requisitos funcionales (RF):

RF-1.– Parsear ficheros de descripción de conjunto de datos.

RF-2.– Permitir el almacenamiento de un conjunto de datos en memoria.

RF-3.– Parsear ficheros de descripción de tabla.

RF-4.– Parsear ficheros de datos.

- **RF-4.1.–** CSV
- **RF-4.2.–** JSON
- **RF-4.3.–** XML
- **RF-4.4.–** RDF
- **RF-4.5.–** XLSX

RF-5.– Almacenar conjuntos de datos en SGBDs.

- **RF-5.1.–** Almacenar conjuntos de datos en MariaDB.
- **RF-5.2.–** Almacenar conjuntos de datos en Elasticsearch.
- **RF-5.3.–** Crear automáticamente index patterns en Kibana.

RF-6.– Borrar conjuntos de datos de SGBDs.

- **RF-6.1.–** Borrar conjuntos de datos de MariaDB.
- **RF-6.2.–** Borrar conjuntos de datos de Elasticsearch.
- **RF-6.3.–** Eliminar automáticamente index patterns de Kibana.

Además, cuenta con estos no funcionales (RNF):

RNF-1.– Eficiencia.

RNF-2.– Versatilidad.

RNF-3.– Usabilidad.

RNF-4.– Escalabilidad.

RNF-5.– Sistema de logging.

3.2. Arquitectura

Balloon posee la arquitectura presentada en la figura 3.1.

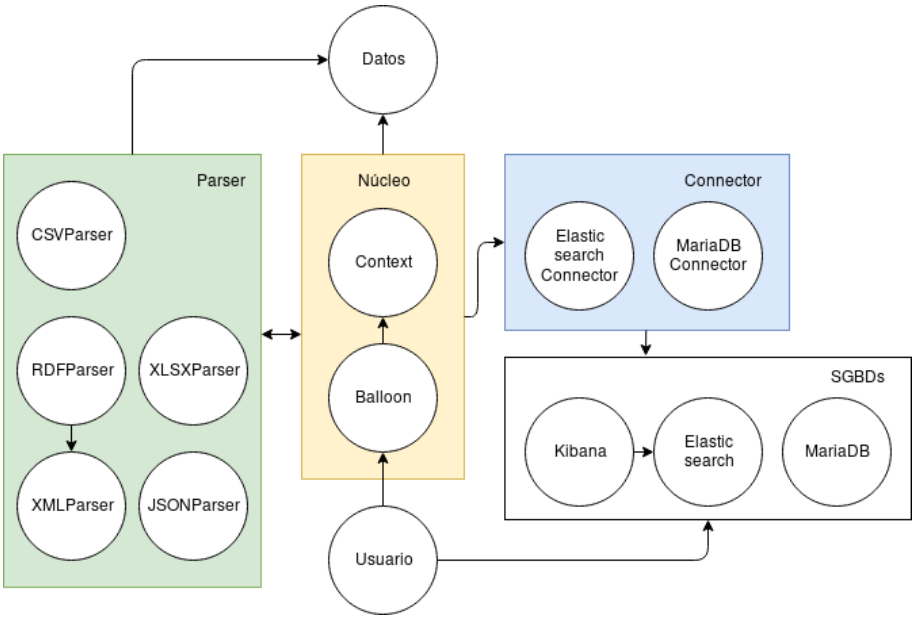

Figura 3.1: Arquitectura de Balloon.

Dada la naturaleza orientada a objectos del sistema, se decide hacer uso de Java. Concretamente, de la versión 8, que es la que recomienda Elasticsearch [19]. Para la conexión de Balloon con MariaDB se emplea su conector JDBC y para la conexión con Elasticsearch, su cliente REST de alto nivel de Java. La comunicación con Kibana se produce a través de su API REST, vía cURL.

El usuario llama a Balloon especificando una acción a aplicar sobre un conjunto de datos. Con esta información se genera un contexto de traducción (Context). Este contexto llama a los parsers que correspondan según el formato de cada fichero. Además, el contexto inserta los datos en un SGBD empleando un conector o simplemente los carga en memoria, según lo que el usuario haya especificado.

Los formatos soportados actualmente son CSV, JSON, XML, RDF y XLSX, mientras que los SGBDs disponibles son Elasticsearch y MariaDB.

Los parsers implementan una interfaz llamada *Parser* y los conectores otra, llamada *Connector*. Con esto se pretende que la adición de componentes de ambos tipos sea especialmente sencilla, cogiendo inspiración del concepto de plugin.

La idea es que la mayor parte del proceso de traducción sea transparente al usuario. Lo único que el usuario debe especificar es la definición de cada tabla, para que Balloon sepa cómo interpretarla. El usuario debe incluir estos ficheros de descripción en el directorio del conjunto de datos, llamar al programa y los datos se importan en el destino deseado, con la consecuente creación automática

de tablas, comprobación de duplicados, empleo de tipos nativos, generación de índices y evasión de caracteres no permitidos.

En base a la figura 3.1 se puede dividir Balloon en tres partes:

- Núcleo.
- Parsers.
- Conectores.

A continuación, se explican en detalle los ficheros de descripción, componente clave de Balloon.

3.3. Ficheros de descripción

En Balloon existen dos tipos de ficheros de descripción:

- Los de conjunto de datos, descritos en el próximo apartado 3.3.1.
- Los de tabla, descritos en el próximo apartado 3.3.2.

En ambos casos se emplea una estructura XML. Para los ficheros de conjunto de datos se usa la extensión *dcat*, mientras que para los de tabla se emplea la extensión *desc*. Estas extensiones son importantes ya que Balloon las tiene en cuenta para diferenciar unos ficheros de otros.

Todos los ficheros de descripción han de colocarse en un mismo directorio en la raíz del directorio de datos. El nombre de esta carpeta es el que obtiene el conjunto de datos en el sistema. Un ejemplo de estructura es el siguiente:

```
balloon
└─data
  ├─transport
  │ ├─transport.dcat
  │ ├─calendar.desc
  │ ├─calendar.xlsx
  │ ├─transport_lines.desc
  │ └─transport_lines.json
  └─dog_areas
    ├─dog_areas.desc
    └─dog_areas.csv
```

Como se puede observar en el esquema, existen dos conjuntos de datos (*transport* y *dog_areas*). *transport* contiene un fichero de descripción de conjunto de datos (DCAT) y dos tablas (*calendar* y *transport_lines*). Cada una de estas tablas cuenta con un fichero de datos (XLSX y JSON, en este caso) y un fichero de descripción (con extensión *.desc*). El conjunto *dog_areas*, por el contrario, no tiene un fichero de descripción de conjunto, ya que es opcional. Contiene la tabla *dog_areas*, es decir, el fichero de datos y el de descripción asociado.

3.3.1. Conjunto de datos

El ejemplo del código 3.1 muestra de manera simplificada el tipo de fichero RDF que publican los Datos Abiertos de Madrid.

Código 3.1: Ejemplo simplificado de fichero de descripción de conjunto de datos, perteneciente al conjunto de Decide Madrid.

```
1  <rdf:RDF>
2      <dcat:Catalog>
3          <dcat:dataset>
4              <dcat:Dataset rdf:about="https://datos.madrid.es/egob/catalogo/300114-0-madrid-decide">
5                  <dct:title>Participación ciudadana. Debates y propuestas</dct:title>
6                  <dct:description>Información de debates y propuestas ciudadanas que figuran en la
                       plataforma de participación ciudadana http://decide.madrid.es, así como información
                       de comentarios y apoyos de los anteriores, e información auxiliar
                       necesaria.></dct:description>
7                  <dcat:theme
                       rdf:resource="http://datos.gob.es/kos/sector-publico/sector/sociedad-bienestar" />
8                  <dct:modified>2017-05-04T00:00:00</dct:modified>
9              </dcat:Dataset>
10         </dcat:dataset>
11     </dcat:Catalog>
12 </rdf:RDF>
```

Los campos que busca Balloon dentro del fichero de descripción para definir un conjunto de datos son exactamente los que aparecen en el ejemplo. El fichero de descripción real contiene muchos más campos pero, al ser ignorados por Balloon, se excluyen por motivos de claridad. En el caso de querer cambiar las referencias a las etiquetas se puede hacer fácilmente en la parte de declaración de atributos de Context.java.

3.3.2. Tabla

El fichero de descripción de tabla presente en el código 3.2 pertenece a las propuestas del conjunto de Decide Madrid.

Los ficheros de descripción de tabla se asocian a los de datos por el nombre. Es decir, se han de llamar igual, excepto por la extensión. En el caso de que el fichero de datos no se encuentre presente, Balloon lo descarga automáticamente haciendo uso del campo *url* de la descripción y lo convierte a UTF-8, en caso de ser necesario. Si el fichero de datos es añadido manualmente, hay que cerciorarse de que la codificación sea UTF-8, para que la visualización de caracteres no estándar sea satisfactoria.

El contenido ha de ser el que se muestra en el código 3.3.

Para <table>:

Código 3.2: Ejemplo de fichero de descripción de tabla

```
<table url="https://datos.madrid.es/egob/catalogo/300114-10434614-madrid-decide.csv"
        props="intrabreaks,return,quotes">

    <attr name="id" props="pk"/>
    <attr name="title"/>
    <attr name="description"/>
    <attr name="cached_votes_up" type="lng"/>
    <attr name="comments_count"type="lng"/>
    <attr name="hot_score" type="itg"/>
    <attr name="confidence_score" type="itg"/>
    <attr name="created_at" type="dte" props="timeref"/>
    <attr name="summary"/>
    <attr name="video_url"/>
    <attr name="geozone_id" type="itg"/>
    <attr name="retired_at" type="dte"/>
    <attr name="retired_reason"/>
    <attr name="retired_explanation"/>
    <attr name="proceeding"/>
    <attr name="sub_proceeding"/>

</table>
```

Código 3.3: Contenido del fichero de descripción de tabla.

```
<table [url="..."] [desc="..."] [base="..."] [props="..."]>
    <attr name="..." [type="..."] [desc="..."] [props="..."]/>
</table>
```

- url: la localización en Internet del fichero de datos.
- desc: una descripción.
- base: para el caso de JSON, XML y RDF, el nombre del objeto que define un registro. En el caso de CSV y XLSX cada línea es un registro.
- props: banderas de descripción del fichero de datos para CSV:
  - quotes: si los valores están englobados por comillas dobles.
  - return: si en los saltos de línea se emplea retorno de carro.
  - intrabreaks: indica que existen saltos de línea dentro de los valores.

Para <attr>:

- name: el nombre del atributo.
- type: el tipo. Por defecto es *str*.
- desc: una descripción.
- props: banderas de descripción:
  - pk: si es clave primaria.
  - lat: si representa una latitud geográfica.
  - lon: si representa una longitud geográfica.
  - timeref: si ha de considerarse como la referencia temporal del registro.
  - decimalcomma: si emplea comas en los números con componentes decimales, en lugar de puntos.

Los tipos de datos que emplea Balloon y sus equivalentes en MariaDB y Elasticsearch son los que se presentan en la tabla 3.1.

Balloon	MariaDB	Elasticsearch
str	TEXT	text
itg	INTEGER	integer
lng	BIGINT	long
dbl	DOUBLE	double
boo	TINYINT	boolean
dte	DATETIME	date
tim	TIME	text

Tabla 3.1: Tipos en Balloon y su equivalencia con MariaDB y Elasticsearch.

El nombre de los atributos es importante. Ha de coincidir con el que aparece en el fichero de datos para que el parser pueda encontrarlo. En el caso de atributos dentro de otros, la nomenclatura es del tipo *padre.hijo*, siendo *padre* e *hijo* el nombre de los respectivos atributos. Esta nomenclatura se puede extender a más padres. Por ejemplo, con la estructura del código 3.4, el nombre de *atrib3* debería ser *atrib1.atrib2.atrib3*.

Con el fichero del código 3.2 Balloon es capaz de realizar automáticamente todo el procesamiento

Código 3.4: Ejemplo de jerarquía de atributos.

```
{
  "atrib1": {
    "atrib2": {
      "atrib3": "valor"
}}}
```

hasta que los datos aparecen en el SGBD correspondiente, como puede apreciarse en las figuras 3.2, 3.3 y 3.4.

```
MariaDB [balloon]> describe proposals;
+---------------------+-------------+------+-----+---------+-------+
| Field               | Type        | Null | Key | Default | Extra |
+---------------------+-------------+------+-----+---------+-------+
| id                  | varchar(20) | NO   | PRI | NULL    |       |
| title               | text        | YES  |     | NULL    |       |
| description         | text        | YES  |     | NULL    |       |
| cached_votes_up     | bigint(20)  | YES  |     | NULL    |       |
| comments_count      | bigint(20)  | YES  |     | NULL    |       |
| hot_score           | int(11)     | YES  |     | NULL    |       |
| confidence_score    | int(11)     | YES  |     | NULL    |       |
| created_at          | datetime    | YES  |     | NULL    |       |
| summary             | text        | YES  |     | NULL    |       |
| video_url           | text        | YES  |     | NULL    |       |
| geozone_id          | int(11)     | YES  |     | NULL    |       |
| retired_at          | datetime    | YES  |     | NULL    |       |
| retired_reason      | text        | YES  |     | NULL    |       |
| retired_explanation | text        | YES  |     | NULL    |       |
| proceeding          | text        | YES  |     | NULL    |       |
| sub_proceeding      | text        | YES  |     | NULL    |       |
+---------------------+-------------+------+-----+---------+-------+
16 rows in set (0.001 sec)
```

Figura 3.2: Tabla de propuestas en MariaDB.

3.4. Organización

Se ha organizado el proyecto de manera que cada parte se encuentra localizada en un directorio independiente. Existen los siguientes cinco:

- conf: carpeta en la que se colocan los ficheros de configuración.
- data: hogar de los conjuntos de datos.
- log: directorio donde se encuentran los logs. Actualmente se escriben dos ficheros: el de ejecución y el de test.
- scripts: en este lugar se incluyen scripts auxiliares a Balloon.
- src: emplazamiento del código fuente. Dentro de esta carpeta se puede encontrar el directorio main, donde se incluye el código de Balloon y la carpeta test.

```
MariaDB [balloon]> select * from proposals where id = 10\G ;
*************************** 1. row ***************************
                id: 10
             title: Hagamos que los madrileños reciclen con confianza
       description: <p>Ya que mucha gente sigue poniendo de excusa la p
oca transparencia en los procesos de reciclaje para no reciclar  (el típ
ico ""da igual, si al final todo va al mismo sitio""); yo digo que les h
agamos ver que no, que su esfuerzo cuenta y que están contribuyendo a qu
e el dia de mañana nuestra ciudad sea algo más que un basurero. Es más,
que nuestra ciudad sea un ejemplo de limpieza y civismo. </p>
    cached_votes_up: 317
     comments_count: 6
          hot_score: 0
   confidence_score: 31700
         created_at: 2015-09-15 10:00:00
            summary: Propongo que desde el ayuntamiento de Madrid se hag
a un video promocional de cómo se llevan a cabo los procesos de reciclaj
e en nuestra comunidad.
          video_url:
        geozone_id: 0
        retired_at: NULL
     retired_reason:
retired_explanation:
        proceeding:
     sub_proceeding:
1 row in set (0.031 sec)
```

Figura 3.3: Ejemplo de propuesta en MariaDB.

```
 7 ▾  "_source": {
 8      "summary": "Propongo que desde el ayuntamiento de Madrid se haga un video
            promocional de cómo se llevan a cabo los procesos de reciclaje en nuestra
            comunidad.",
 9      "hot_score": 0,
10      "retired_explanation": "",
11      "description": "<p>Ya que mucha gente sigue poniendo de excusa la poca
            transparencia en los procesos de reciclaje para no reciclar··(el típico
            \\\"\\\"da igual, si al final todo va al mismo sitio\\\"\\\");·yo digo que
            les hagamos ver que no, que su esfuerzo cuenta y que están contribuyendo a
            que el dia de mañana nuestra ciudad sea algo·más que un basurero. Es más,
            que nuestra ciudad sea un ejemplo de limpieza y civismo.·</p>",
12      "created_at": "2015-09-15 10:00:00",
13      "proceeding": "",
14      "title": "Hagamos que los madrileños reciclen con confianza",
15      "cached_votes_up": 317,
16      "sub_proceeding": "",
17      "video_url": "",
18      "geozone_id": 0,
19      "comments_count": 6,
20      "confidence_score": 31700,
21      "retired_reason": "",
22      "id": "10"
23  },
```

Figura 3.4: Ejemplo de propuesta en Kibana.

El software encargado de orquestrar todos los procesos de desarrollo es Maven. Este programa se encarga de resolver y descargar las dependencias, compilar el código, ejecutar los tests, generar el Javadoc y empaquetar el programa en un JAR. Además, se desarrolló un Makefile para simplificar aún más los comandos a ejecutar por el desarrollador. Este Makefile llama a los comandos de Maven.

El paquete principal del código se llama *balloon*. En este paquete se encuentra el núcleo (*Balloon* y *Context*). Existen tres subpaquetes:

- balloon.parsers: los parsers y la interfaz *Parser*.
- balloon.connectors: los conectores y la interfaz *Connector*.
- balloon.entities: los componentes necesarios para mantener un conjunto de datos en memoria.

El objetivo de esta distribución de carpetas y paquetes es aumentar la escalabilidad.

4

Implementación

La documentación de la API se puede generar ejecutando *make doc*. El objetivo de las siguientes secciones es describir la funcionalidad de cada integrante del programa tal y como se desglosa en la figura 3.1.

4.1. Balloon

Es la clase principal del programa, que recibe los argumentos del usuario.

En su sección de atributos se pueden encontrar los valores de ejecución por defecto. Son estos mismos los que el usuario puede sobreescribir a su antojo. Lo primero que ocurre en el main es exactamente esto. Se definen los argumentos de entrada haciendo uso de *java-getopt*, estableciendo asociaciones entre argumentos cortos, del estilo -f (Getopt), y largos, del tipo --flag (LongOpt). Esto ofrece versatilidad a la hora de llamar al programa y aumenta la usabilidad (RNF-2 y RNF-3).

Después de la actualización de los valores por defecto se inicializa el logger raíz con *Log4j 2*. Se decidió emplear un sistema de logging para dar la opción al usuario de elegir cuánta información y de qué tipo mostrar por pantalla y para persistir esta información en ficheros. Se podría haber hecho con condicionales, impresiones a stdout y escribiendo manualmente en ficheros pero *Log4j 2* simplifica todo este proceso y permite mayor escalabilidad (RNF-4 y RNF-5).

En la parte final del main se crea una instancia de *Context* con los valores especificados por el usuario. Entre estos valores se encuentra el conjunto de datos a procesar. Para terminar, se llama al método *action* del contexto, pasándole como argumento el nombre del método que debe ejecutar.

4.2. Context

Es el cerebro de Balloon, que determina qué hay que traducir y cómo.

En la parte de declaración de atributos contiene un número relativamente elevado de componentes.

La razón de esto es evitar “cadenas mágicas” en los métodos. Se podrían haber centralizado todas las constantes en *Balloon.java*, pero afectaría a la simplicidad de su clase. Se dividió esta parte de declaraciones por categoría:

1.– Constantes comunes a todos los componentes de Balloon como definición de tipos y delimitadores.

2.– Constantes referentes a los elementos a encontrar en los ficheros de descripción de conjunto de datos.

3.– Constantes y variables relacionadas con la etiqueta <table> de los ficheros de descripción de tabla.

4.– Constantes relacionadas con la etiqueta <attr> de los ficheros de descripción de tabla.

5.– Atributos privados.

En el comienzo de la parte de declaración de métodos se encuentran tres constructores. Aunque esto incrementa el número de líneas de código, resulta en un aumento de la versatilidad y evita la aparición de llamadas al constructor con demasiados null. Seguidamente se encuentran los getters, los setters y la zona de métodos auxiliares.

En este momento aparecen los métodos verdaderamente importantes:

4.2.1. loadFileNames()

Este método accede al directorio con el mismo nombre que el conjunto de datos especificado por el usuario y lista todos los ficheros, comprobando la extensión de cada uno en un bucle. El nombre del fichero de descripción de conjunto de datos (*.dcat) lo guarda en la variable *datasetDescFileName*. Cualquier nombre de fichero de descripción de tabla lo añade a la lista *descFileNames* y cualquier nombre de fichero de datos lo almacena en *dataFileNames*. En el caso de que la acción del contexto sea *add*, itera sobre la lista de nombres de ficheros de descripción de tabla comprobando que el fichero de datos asociado se encuentra presente en *dataFileNames*. Si no lo está, lo descarga y convierte, si es necesario, a UTF-8, renombrándolo para que el nombre coincida con el del fichero de descripción (exceptuando la extensión) y añadiendo este nombre a *dataFileNames*. Finalmente, ordena las dos listas alfabéticamente para que en el mismo índice se sitúen los ficheros asociados.

4.2.2. readDatasetDesc()

Haciendo uso de DocumentBuilder parsea el fichero de descripción de conjunto de datos. Para ello se vale de la constante *DATASET_BASE*, cuyo valor es el nombre del objeto que contiene la información del conjunto de datos en la estructura XML. Extrayendo el valor de los atributos correspondientes a la URL, la descripción, las categorías y la fecha de última modificación genera una instancia de *Dataset* (sin tablas), implementándose el RF-1.

4.2.3. readTableDesc()

El funcionamiento es análogo al de *readDatasetDesc*, aunque la situación es más complicada, por incluir un bucle. Emplea la misma clase para parsear el XML. Extrae todos los atributos de la etiqueta <table> y después obtiene la lista de <attr>, la cual itera, extrayendo el valor de sus atributos. Al final de cada iteración genera una instancia de tipo *DataAttribute*, que incluye en la lista de atributos del contexto (RF-3).

4.2.4. createRecord()

Este es un método clave. Es al que llaman los parsers, pasándole una lista de cadenas de texto con el valor de cada atributo de un registro. Este método convierte cada valor a su tipo correspondiente, valiéndose de la lista de atributos generada en *readTableDesc*. Es importante que el orden en el que aparecen los atributos en el fichero de descripción de tabla coincida con el orden de parseo, para que la conversión de tipos tenga sentido. Para garantizar esto, se emplean listas de tipo java.util.List, que garantizan el orden. En este método hay gran cantidad de pequeñas circunstancias que se toman en cuenta para la conversión de tipos y que no se entra a discutir.

Al principio de este método se instancia un *DataRecord* vacío y cada valor parseado se le añade, por lo que al final del bucle se tiene un DataRecord perfectamente formado. *createRecord* también se encarga de insertar en la base de datos que corresponda. Si el contexto tiene un objeto inicializado de tipo *Connector*, llama a su método *insert* con el registro.

En el caso de que se haya insertado el registro en una base de datos, *createRecord* devuelve null. En caso contrario, se entiende que el usuario desea cargar el registro en memoria, y se devuelve. El motivo de esto es no sobrecargar la memoria innecesariamente.

4.2.5. add()

Este método se encarga de añadir un conjunto de datos al destino especificado por el usuario. Inicialmente, comprueba si el deseo del usuario es cargar el conjunto de datos en memoria. En ese caso lee el fichero de descripción de conjunto de datos. En caso contrario, no lo hace, ya que los metadatos del conjunto no son útiles ni en MariaDB ni en Elasticsearch.

Seguidamente itera sobre la lista de nombres de fichero de descripción de tabla. En cada iteración parsea el fichero de descripción, llama al método *createTable* del conector, en caso de ser necesario y llama al parser correspondiente con el fichero de datos asociado. Esta llamada es, al igual que la llamada a la acción del contexto del apartado 4.1, dinámica, para promover la escalabilidad (véase 4.2.7). La restricción que esto incluye es que el nombre de los parsers ha de empezar con las siglas en mayúscula de la extensión que procesan, seguido de 'Parser'. Por ejemplo, el parser de ficheros .csv

ha de llamarse necesariamente CSVParser. Por ende, si deseamos añadir un nuevo parser a Balloon, basta con crear la nueva clase siguiendo la nomenclatura establecida, y el código en Context no hace falta modificarlo.

4.2.6. remove()

La situación es análoga a la de *add*, pero más sencilla porque no hay parseo. Itera también sobre la lista de nombres de fichero de descripción de tablas, de los que se vale para determinar el nombre de la tabla a borrar en cada iteración. Para borrarla llama al método *deleteTable* del conector (RF-6.1 y RF-6.2). Como con el borrado de la tabla se efectúa el borrado de todo su contenido, no hace falta hacer nada más. Al final del método se comprueba si el borrado es fuerte. En caso de ser así, se elimina también el fichero de datos asociado.

4.2.7. action()

Inicializa el conector correspondiente, en caso de que el usuario desee importar los datos en un SGBD. Obtiene dinámicamente el método a ejecutar en función del nombre pasado como argumento por Balloon y comprueba si el usuario ha especificado un conjunto de datos determinado. En caso de ser así, ejecuta la acción que corresponda sobre ese conjunto. En caso contrario, procesa todo el contenido del directorio de datos. Al final del proceso cierra el conector abierto.

El propio Balloon podría en el main, haciendo uso de un condicional, llamar directamente al método correspondiente en el contexto, pero se decidió hacer de forma dinámica por los siguientes motivos:

- Simplifica el código en el main y el sobrecoste en Context es mínimo.
- Hace más sencillo incluir futuras acciones en Balloon.

4.3. Parsers

Se trató de que los parsers fueran lo más sencillos posible, buscando la máxima independencia con respecto al programa y la mejor integración posible. Para ello, se definió la interfaz Parser con un sólo método: *parse*. De esta manera, el contexto considera a todos los parsers lógicamente iguales.

4.3.1. Parsers por línea

Estos parsers consideran que cada línea del fichero es un registro.

CSVParser (RF-4.1) lee un fichero, lo parte según las especificaciones del usuario (hay variantes

de CSV) para obtener los registros e itera sobre todos estos, dividiéndolos en sus valores (otra vez según las especificaciones del usuario). Para estas divisiones se emplea el método split de String. Si falta cualquier valor, se rellena con un valor por defecto. Con cada una de estas listas de valores llama al método *createRecord* del contexto para generar el *DataRecord*, ya que es el contexto el que sabe qué tipo de dato corresponde a qué valor y el que se encarga de convertir al tipo correspondiente. En caso de que *createRecord* devuelva un registro, se inserta en una lista. Una vez traducido todo el archivo, el traductor devuelve la lista de registros. Si los datos no se están cargando en memoria, en la lista de registros no se habrá incluido ninguno, por lo que se devuelve vacía.

XLSXParser (RF-4.5) emplea la librería POI de Apache, que lee el fichero haciendo uso de la WorkbookFactory. Se obtiene un iterador de filas que se puede ir recorriendo de manera análoga a CSVParser.

4.3.2. Parsers por objeto

Estos parsers consideran que un objeto en el fichero corresponde a un registro.

JSONParser (RF-4.2) hace uso de JSON In Java para parsear el fichero. Obtiene una lista de objetos que representan los registros y cuyo nombre es el especificado por el usuario en el fichero de descripción. Itera sobre esta lista, obteniendo el valor de cada atributo.

Para encontrar el atributo deseado dentro del objeto se vale del nombre del primero. Esta búsqueda es recursiva y está implementada en un método auxiliar privado llamado *getValue*. Éste método lo poseen todos los parsers por objeto, cada uno con su implementación. Una vez recogido el valor de todos los atributos llama a *createRecord* con la lista, de manera análoga a *CSVParser*. El resto del proceso es el mismo.

XMLParser (RF-4.3) emplea DocumentBuilder para parsear el fichero.

RDFParser (RF-4.4) hereda de *XMLParser*, ya que que RDF es una implementación de XML. XMLParser y RDFParser, por lo tanto, hacen lo mismo.

4.4. Conectores

El enfoque es similar al de los parsers: se definió la interfaz *Connector*, que deben implementar todos los conectores. Esta interfaz cuenta con seis métodos:

- createTable
- insert: en cada llamada a este método se añade el registro a un lote, que cuando alcanza un máximo determinado se manda al SGBD, para aumentar la eficiencia (RNF-1).
- closeBatch: se encarga de mandar la parte remanente del lote y cerrarlo.

- select: devuelve un objeto con el registro presente en la tabla especificada como argumento y que cumple la condición también especificada.
- deleteTable
- close: cierra la conexión con el SGBD.

4.4.1. MariaDBConnector

Implementa el RF-5.1. Hace uso del conector JDBC de MariaDB, según el protocolo *jdbc:mariadb*. Al inicializarse, parsea el fichero XML de configuración de conexión (dirección y credenciales) presente en el directorio *conf*, información de la que se vale para establecer la conexión.

En resumidas cuentas, lo que hace este conector es recibir una orden del contexto, traducirla a SQL y llamar a los métodos de la clase *Connection* de java.sql para actualizar la base de datos.

La manera que tiene de saber a qué tipo de MariaDB ha de convertir un tipo de Balloon es consultando la lista MARIADB_TYPES del contexto. Cabe destacar que cualquier nombre que entre en MariaDB pasa por el método *beautify* del contexto para eliminar caracteres no permitidos. Este método se encuentra en *Context* ya que se aplica la misma nomenclatura a los nombres que se insertan en Elasticsearch para que sean siempre iguales. Con esto se pretende alcanzar una cierta consistencia.

Los lotes se abren nada más crear la tabla y los ha de cerrar el contexto cuando determine que ya no hay más registros que insertar. Se implementan haciendo uso de la clase *Statement* de java.sql, que añade las consultas de inserción a un buffer valiéndose del método addBatch. Cuando se alcanza el tamaño máximo de lote, se mandan a MariaDB con su método executeBatch y se abre un nuevo lote.

El método *select* devuelve un *DataTable* con el resultado de la consulta a MariaDB.

4.4.2. ElasticsearchConnector

Implementa el RF-5.2. En realidad este conector no sólo gestiona la comunicación con Elasticsearch, sino que también la gestiona con Kibana. Se incluyó esta funcionalidad aquí por su limitado tamaño y su estrecha relación con Elasticsearch.

La clase empleada para mandar las consultas es el RestHighLevelClient de Elasticsearch. La dirección de conexión la obtiene de Balloon, ya que el usuario es libre de modificarla.

Se han empleado mapas para construir las consultas en lugar de cadenas de texto, ya que simplifica mucho el desarrollo. A la hora de crear un índice se genera un mapping valiéndose de la lista de atributos del contexto. La manera de consultar los tipos es la misma que la que emplea *MariaDBConnector*. En este caso, el lote también se inicializa nada más crear el índice, tras lo cual se manda un POST a la API de Kibana, especificando la creación de un index pattern con el mismo nombre (RF-5.3).

Con esto se aumenta la automatización.

Tanto en el caso de Elasticsearch como en el de Kibana se tienen en cuenta algunos atributos propios de estas tecnologías especificadas por el usuario en el fichero de descripción de tabla, como el *timeref* (referencia temporal de Kibana) y el *lat*/*lon* (necesarios para definir una posición geográfica en Elasticsearch).

El sistema de lotes es análogo al de MariaDB. En el caso del método *select*, en lugar de devolver un *DataTable*, devuelve un JSONObject. Con el borrado de un índice se borra también el index pattern asociado (RF-6.3).

4.5. Entidades

El objetivo de este grupo es definir los componentes básicos de un conjunto de datos:

- Dataset
- DataTable
- DataRecord
- DataAttribute

Estas clases están relacionadas como dicta la intuición. Es este esquema el empleado cuando se carga un conjunto de datos en memoria y, en el caso de MariaDB, cuando se construye el resultado de un *select* (RF-2).

5 Evaluación

5.1. Pruebas

Para verificar el correcto funcionamiento de Balloon se empleó JUnit, con 7 pruebas unitarias (5 parsers y 2 conectores) y 3 pruebas de integración (una por cada motor de procesamiento). La manera ideal de ejecutar el test es lanzando *make test*.

En cada una de las pruebas unitarias de los parsers se llamó su método *parse* con un fichero de datos simplificado, comprobando que los registros generados eran iguales a los esperados.

En cada una de las pruebas unitarias de los conectores se creó una tabla, se añadió un registro y se realizó una consulta que devolviera el registro, comprobándose que sus valores eran iguales a los esperados. Finalmente, se borró la tabla creada y se comprobó la efectividad de esta acción con otra consulta al SGBD correspondiente.

En las tres pruebas de integración (memoria, MariaDB y Elasticsearch) se procesó un conjunto de datos completo basado en datos simplificados de conjuntos reales de los datos abiertos del Ayuntamiento de Madrid. Concretamente, las tablas simplificadas del conjunto procesado fueron:

- test_accidents_2010.xlsx
- test_activities.xml
- test_cultural_events.json
- test_library_events.rdf
- test_proposals.csv

Cada una de estas tablas es un subconjunto de 10 registros de las tablas originales, para hacer los tests ligeros. Nótese que cada uno de los formatos soportados por Balloon se encuentra representado en uno de los ficheros. El procesamiento consistió en cargar el conjunto en el motor correspondiente, realizar consultas para comprobar la consistencia de los datos y, en el caso de MariaDB y Elasticsearch, borrarlo.

Se trató de mezclar consultas a datos delicados (fechas o coordenadas) con consultas a datos sencillos (texto) para garantizar el buen funcionamiento en todos los casos.

5.2. Caso de uso

Decide Madrid [20] es la plataforma electrónica de presupuestos participativos de la ciudad. En este lugar, los ciudadanos debaten cuestiones referentes a Madrid y realizan propuestas fundamentadas de asignación de recursos presupuestarios. Cada una de estas propuestas va asociada a uno de los 21 distritos de la ciudad o a la ciudad en su conjunto, y su contenido es el que aparece en la figura 3.2. La figura 5.1 muestra el aspecto de la página web de propuestas del portal.

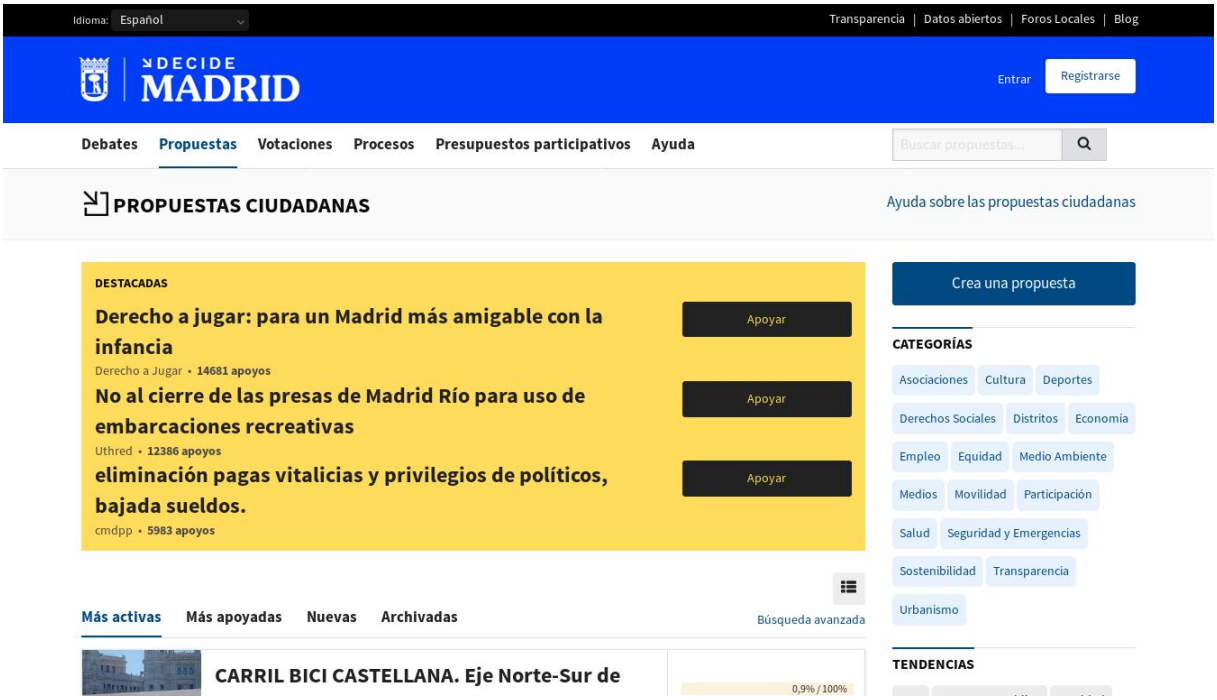

Figura 5.1: Página de propuestas de Decide Madrid.

Antes de realizar el estudio se cargaron los siguientes 15 conjuntos de datos:

- Accidentes de tráfico con implicación de bicicletas (2010-2018) (CSV)
- Accidentes de tráfico de la Ciudad de Madrid (2010-2018) (XLSX)
- Actividades Culturales y de Ocio Municipal en los próximos 100 días (JSON)
- Actividades gratuitas en Bibliotecas Municipales en los próximos 60 días (RDF)
- Actuaciones en materia de limpieza urbana (CSV)
- Agenda de actividades y eventos (XML)
- Áreas caninas (CSV)
- Callejero Oficial del Ayuntamiento de Madrid (CSV)
- Divisiones administrativas: distritos, barrios y divisiones históricas (CSV)
- EMT. Calendario, grupos, líneas y paradas en formato xlsx (XLSX)
- EMT. Incidencias del servicio (XML)
- Participación ciudadana. Debates y propuestas (CSV)
- Taxi. Reservas de paradas en vía pública (XLSX)

- Taxi. Situación y Gestión. Histórico (CSV)
- Tráfico. Incidencias en vía pública (XML)

Además, se combinó la tabla de propuestas de Decide Madrid con la tabla de distritos del mismo conjunto, ejecutando la consulta del código 5.1.

Código 5.1: Combinación de propuestas y distritos en MariaDB.

```
1  CREATE VIEW props_districts AS SELECT proposals.*, name AS geozone_name FROM
       proposals,geozones WHERE geozone_id = geozones.id;
```

Para incluir esta tabla en Elasticsearch se llamó a Logstash con el fichero de configuración del código 5.2.

Código 5.2: Fichero de configuración de Logstash para migrar props_districts.

```
1  input {
2      jdbc {
3          jdbc_connection_string => "jdbc:mariadb://localhost:3306/balloon"
4          jdbc_user => "<user>"
5          jdbc_password => "<password>"
6          jdbc_validate_connection => true
7          jdbc_driver_library => "<path_to_jdbc_driver_library>"
8          jdbc_driver_class => "org.mariadb.jdbc.Driver"
9          statement => "SELECT * from props_districts"
10     }
11 }
12 output {
13     elasticsearch {
14         index => "props_districts"
15         hosts => "localhost"
16     }
17 }
```

Una vez llevadas a cabo estas acciones, se realizó el estudio, que consta de dos partes disjuntas: *Ciclismo seguro* y *Áreas caninas*, las cuales se presentan en los siguientes dos apartados. Estas dos partes constituyen tan sólo una pequeña muestra del gran conjunto de posibilidades que ofrece la integración y el análisis de los datos abiertos; se podrían estudiar indicadores económicos, educación, inserción laboral, contaminación, etc. Es decir, un amplio abanico de temáticas que engloban todos los aspectos de la vida en una ciudad.

El estudio de estos aspectos podría sacar a la luz problemas subyacentes o ayudar a la determinación de situaciones favorables desconocidas. De esta manera, se obtiene un conocimiento muy valioso que facilita la asignación eficiente de recursos, resultando en una mejora de la calidad de vida para

los ciudadanos y en un ahorro económico para el Ayuntamiento. Es por ello que se concluye que el tratamiento de estos datos es beneficioso para todas las partes.

5.2.1. Ciclismo seguro

En esta parte se comprueba si los distritos a los que se asocian la mayoría de propuestas referentes a ciclismo seguro son los mismos en los que se producen más accidentes relacionados con ciclistas.

Para comenzar, se presenta la gráfica de la figura 5.2, que representa el número de propuestas relacionadas con accidentes de ciclismo por mes. En este cómputo se tienen en cuenta tanto las propuestas asociadas a un distrito como las asociadas a Madrid en general. Como se puede observar, durante los primeros meses se publicaron más propuestas que en cualquier otro momento. Una hipótesis de este comportamiento es que Decide Madrid se estrenó en septiembre de 2015, relacionándose el gran número de propuestas con la novedad. Observando la relación entre el número de propuestas publicadas de cualquier tipo con cada mes, presentada en la figura 5.3, se aprecia un comportamiento similar que, además, es parecido en cualquier momento del tiempo. Tanto la figura 5.2 como la 5.3 se generaron a partir de la tabla de propuestas de Decide Madrid, procesada automáticamente por Balloon. Salvando los primeros meses, el número medio de propuestas mensuales relacionadas con el tema es de 10.

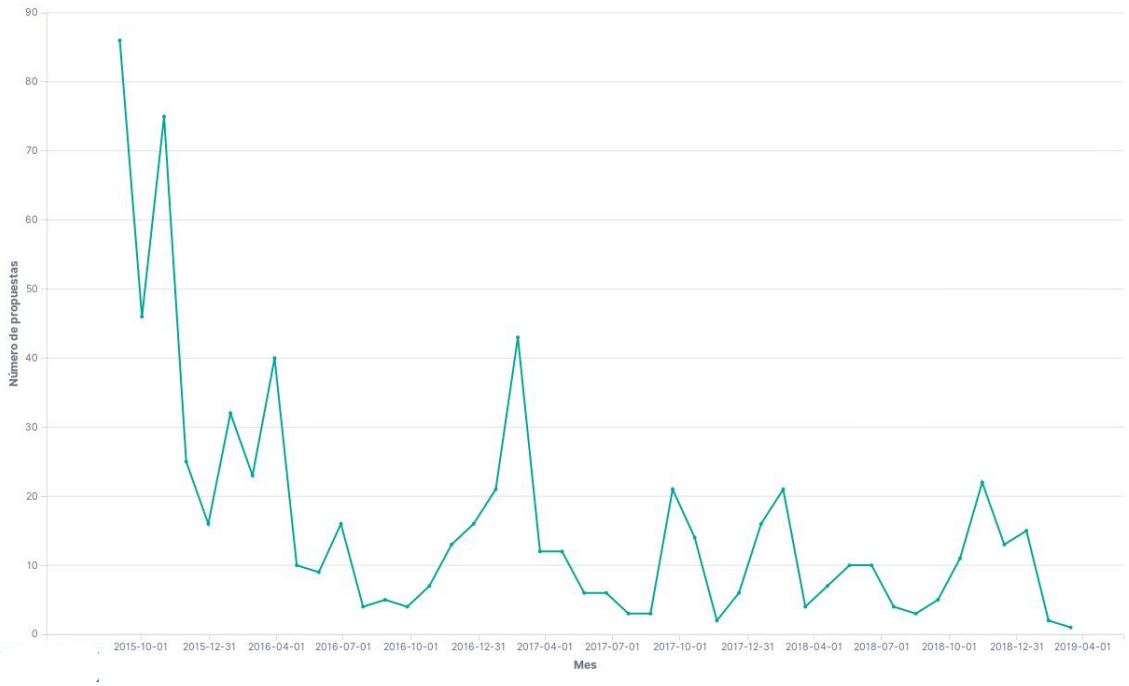

Figura 5.2: Número de propuestas referentes a ciclismo seguro por mes.

Seguidamente, se presentan dos diagramas (figuras 5.4 y 5.5) obtenidos a partir de las siguientes tablas:

Figura 5.3: Número de propuestas por mes.

- Tablas de *Accidentes de tráfico con implicación de bicicletas* entre 2015 y 2018.
- *props_districts* (la combinación de propuestas y distritos de Decide Madrid vista en el apartado anterior).

Tanto la figura 5.2 como la 5.4 se obtienen filtrando en Kibana la tabla de propuestas correspondiente con la cadena del código 5.3.

Código 5.3: Filtro para el diagrama de distribución de propuestas sobre ciclismo seguro por distrito.

```
1 (bici* OR cicli*) AND (accident* OR emergencia* OR peligro* OR segur* OR siniestr* OR choqu* OR
      choc* OR atropell* OR colisi*)
```

Como se puede observar en la figura 5.5, el distrito donde más accidentes de bicicleta se producen (Centro) no es el que se asocia a la mayoría de propuestas relacionadas con el tema (Fuencarral-El Pardo).

5.2.2. Áreas caninas

En esta sección se comprueba si los distritos relacionados con la mayoría de propuestas sobre construcción de áreas caninas son coincidentes con los distritos en los que menos hay. Se comienza con la presentación de una gráfica (figura 5.6) que relaciona el número de propuestas (tanto asociadas a distritos como no) con los meses. Se observa una media de en torno a cinco propuestas mensuales.

En la figura 5.7 se presenta la distribución de propuestas por distrito, mientras que en la figura 5.8 se indica la localización de las áreas actuales. Los diagramas se han confeccionado procesando con Balloon la tabla *props_districts* y el conjunto de *Áreas caninas*.

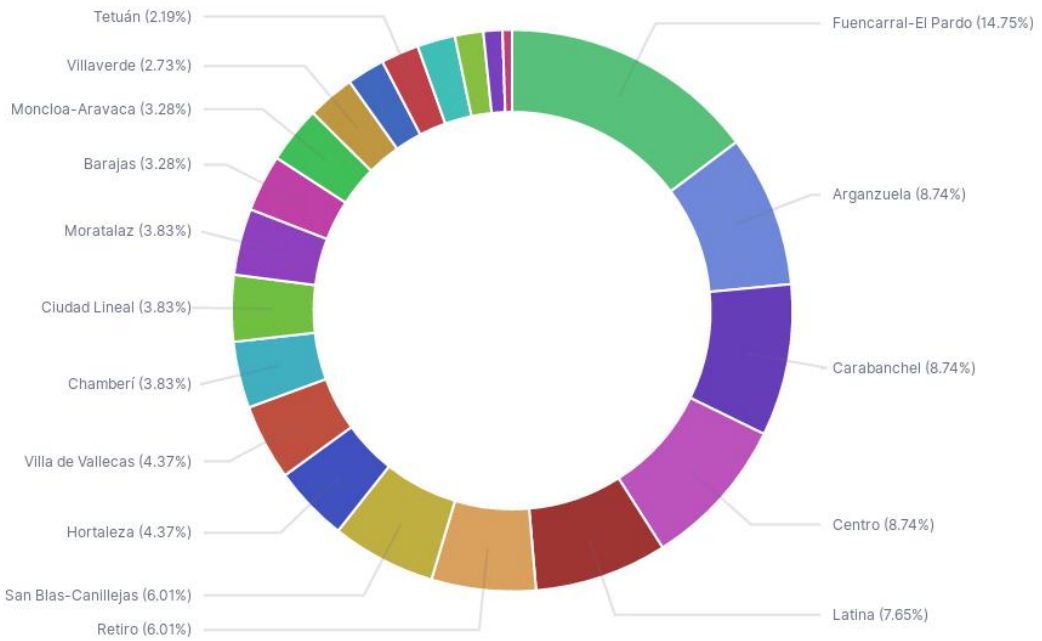

Figura 5.4: Distribución de propuestas sobre ciclismo seguro por distrito.

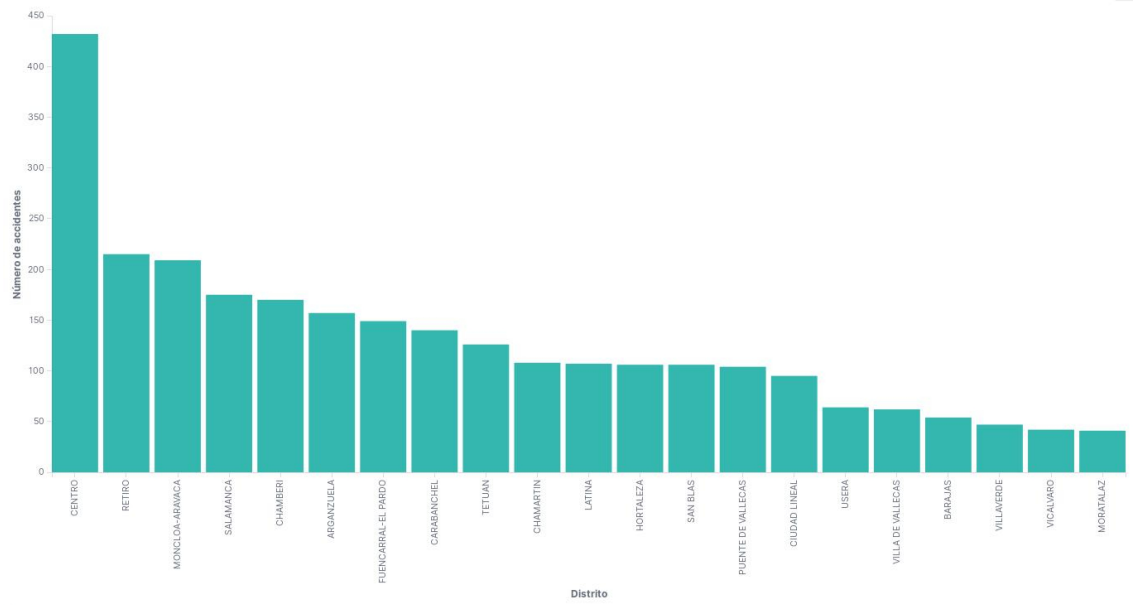

Figura 5.5: Distribución de accidentes de bicicleta entre 2015 y 2018 por distrito.

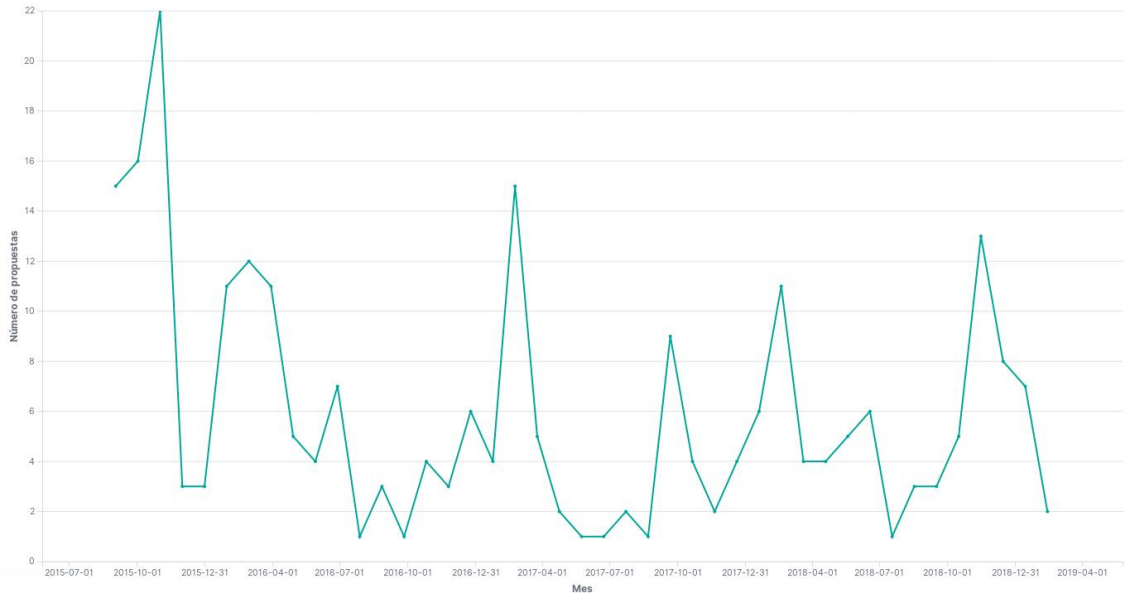

Figura 5.6: Número de propuestas sobre construcción de áreas caninas.

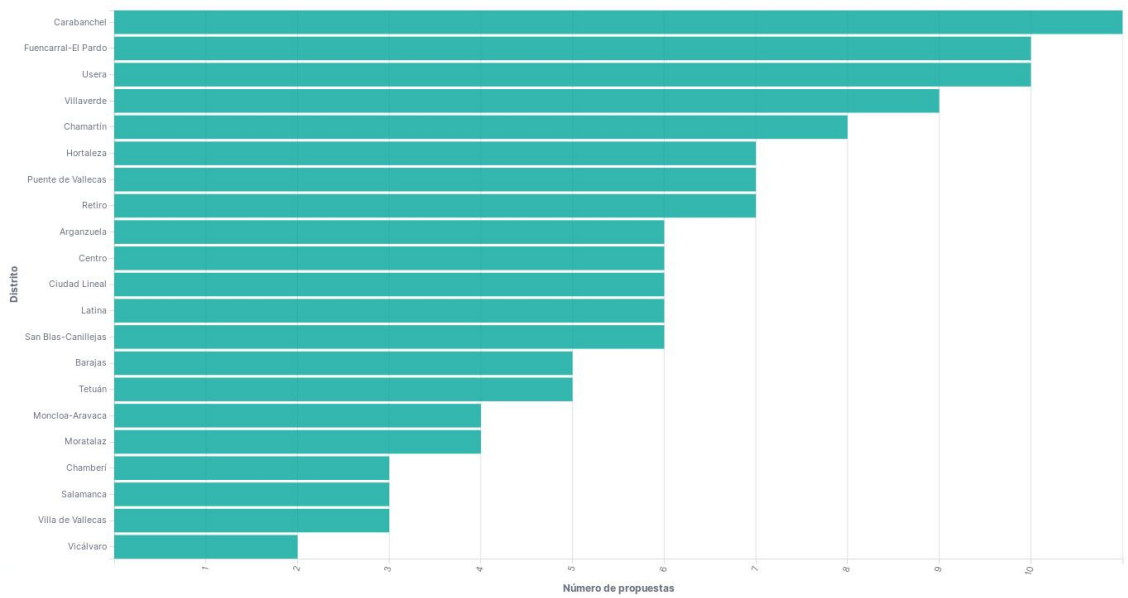

Figura 5.7: Distribución de propuestas sobre construcción de áreas caninas por distrito.

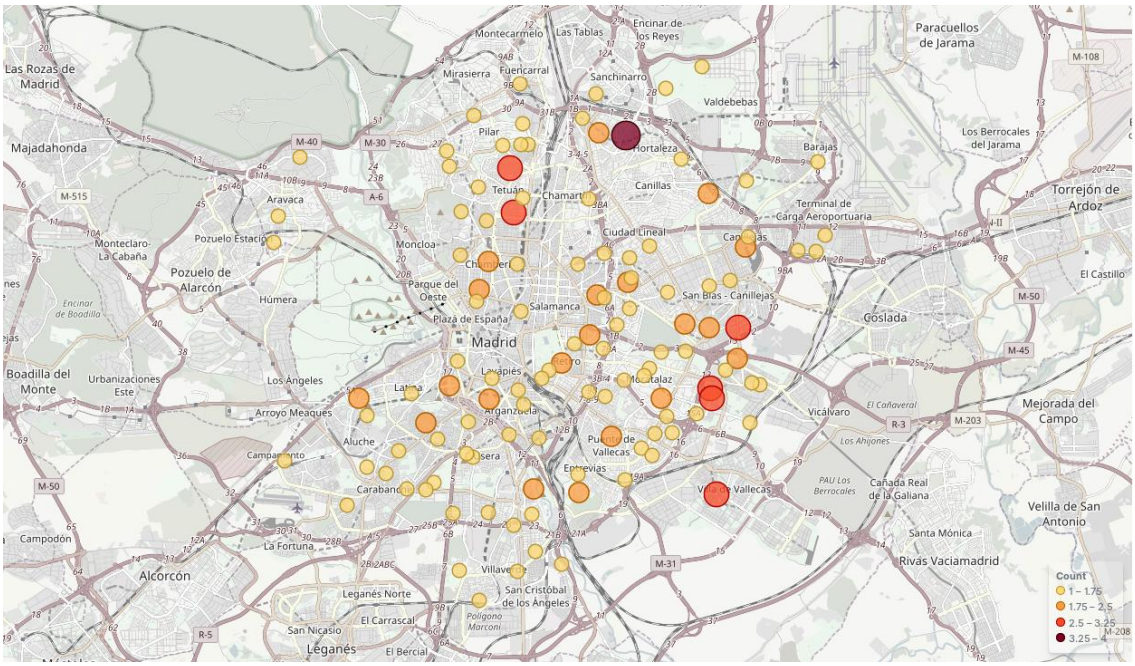

Figura 5.8: Distribución geográfica de áreas caninas.

Tanto la figura 5.6 como la 5.7 se obtienen filtrando la tabla correspondiente con la cadena del código 5.4.

Código 5.4: Filtro para el diagrama de distribución de propuestas sobre construcción de áreas caninas por distrito.

```
1 (zona* OR área* OR parque*) AND (perr* OR canin*) AND (poner OR colocar OR construir OR hacer
      OR incluir)
```

Como se puede observar en la figura 5.8, la distribución de áreas caninas es bastante uniforme en Madrid. Se observa que en Carabanchel ya existen varias áreas caninas, por lo que no se puede concluir que el distrito donde más propuestas se publican relacionadas con el tema es donde menos áreas hay. Sin embargo, en el distrito de Fuencarral-El Pardo no existen demasiadas áreas por el momento (la zona de El Pardo no se ha incluido en el mapa por su vasta extensión y su ausencia de áreas caninas). En este caso, sí se observa una relación entre el número de propuestas publicadas y la ausencia de áreas caninas.

6

Conclusiones

Debido a la gran variedad de datos abiertos que se publican y a la utilización de formatos tan dispares, se reafirma la necesidad de desarrollar aplicaciones especializadas en procesar e integrarlos.

Como se ha podido observar, Balloon se constituye como un sistema que actúa como nexo entre los publicadores de datos abiertos y los SGBDs. Aunque actualmente sólo soporta cinco formatos y dos SGBDs, el objetivo es incrementar este soporte, meta relativamente sencilla debido al enfoque genérico del proyecto.

Balloon cumple con todos los requisitos del proyecto, como se ha anotado en el capítulo *Implementación*. Es capaz de abordar su objetivo principal: automatizar al máximo la carga e integración de los datos. Lo único que el usuario debe hacer es proveer el fichero de descripción de cada tabla.

Durante la elaboración del proyecto se ha observado la complejidad del parseo de datos. Si bien es cierto que los formatos de los conjuntos abiertos suelen ser siempre los mismos, la manera de presentar los datos suele diferir de conjunto en conjunto. Por ejemplo, en algunos ficheros XML el nombre de las etiquetas corresponde al nombre de los atributos, mientras que en otros el nombre de los atributos corresponde al valor de un atributo de la etiqueta, en algunos ficheros CSV los valores se encuentran entrecomillados, en otros existen saltos de línea en medio de un registro... Balloon realiza una labor de abstracción y es capaz de tener todos estos factores en cuenta apenas sin intervención del usuario.

Se observa también que los datos procesados por Balloon son útiles para sacar conclusiones, como las que se presentan en el apartado 5.2. Estas conclusiones son sólo un subconjunto reducido de un gran abanico. Al representar este tipo de datos situaciones reales, su procesamiento tiene repercusiones importantes: se pueden tener en cuenta para tomar decisiones que afectan directamente a la población. Permiten tomar decisiones más fundamentadas.

Todo indica a que la publicación de datos abiertos no hará más que aumentar, por lo que sin duda aparecerán más herramientas como Balloon para enriquecer este ámbito y promover su estudio.

Bibliografía

[1] Open Definition. http://opendefinition.org/.

[2] Open Knowledge International. https://okfn.org/.

[3] Open Definition, "Definición de Conocimiento Abierto." https://opendefinition.org/od/2.1/es/.

[4] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data: The story so far," in *Semantic services, interoperability and web applications: emerging concepts*, pp. 205–227, IGI Global, 2011.

[5] T. Berners-Lee, "Linked Data (Design Issues - W3C)," July 2006.

[6] T. Berners-Lee, J. Hendler, O. Lassila, *et al.*, "The semantic web," *Scientific american*, vol. 284, no. 5, pp. 28–37, 2001.

[7] UNICEF, "UNICEF Data." https://data.unicef.org/.

[8] WWF, "GLOBIL." https://globil-panda.opendata.arcgis.com/.

[9] The World Bank, "World Bank Open Data." https://data.worldbank.org/.

[10] Ayuntamiento de Madrid, "Datos Abiertos de Madrid." https://datos.madrid.es/portal/site/egob/.

[11] Ayuntamiento de Copenhague, "Copenhaguen Data." https://data.kk.dk/.

[12] Ayuntamiento de Estocolmo, "Öppna Data - Dataportalen." https://dataportalen.stockholm.se/dataportalen/.

[13] Costa Rica, "Datos Abiertos." http://datosabiertos.presidencia.go.cr/home.

[14] Rusia, "Datos Abiertos." http://opengovdata.ru/.

[15] EEUU, "DATA.GOV." http://catalog.data.gov/dataset.

[16] Naciones Unidas, "UN Data." http://data.un.org/.

[17] Unión Europea, "EU Open Data Portal." https://data.europa.eu/euodp/en/home.

[18] B. Obama, "Transparency and open government, memorandum for the heads of executive departments and agencies," 2009.

[19] Elastic, "Set up Elasticsearch." https://www.elastic.co/guide/en/elasticsearch/reference/current/setup.html.

[20] Ayuntamiento de Madrid, "Decide Madrid." https://decide.madrid.es/.

Acrónimos

API Application programming interface.

CSV Comma-separated values.

DCAT Data Catalog Vocabulary.

EEUU Estados Unidos.

HTTP Hypertext Transfer Protocol.

JAR Java Archive.

JSON JavaScript Object Notation.

ONG Organización no gubernamental.

POI Poor Obfuscation Implementation.

POM Project Object Model.

RDF Resource Description Framework.

REST Representational State Transfer.

RF Requisito funcional.

RNF Requisito no funcional.

SGBD Sistema de Gestión de Bases de Datos.

SPARQL SPARQL Protocol and RDF Query Language.

SQL Structured Query Language.

TFG Trabajo de Fin de Grado.

UNICEF United Nations Children's Fund.

URI Uniform Resource Identifier.

URL Uniform Ressource Locator.

UTF-8 Unicode Transformation Format 8-bit.

WWF World Wildlife Fund.

XML eXtensible Markup Language.


UAM
UNIVERSIDAD AUTONOMA
DE MADRID